Electronic Notes in Theoretical Computer Science 7 (1997)
URL: http://www.elsevier.nl/locate/entcs/volume7.html ?? pages

Tiles for Concurrent and Located Calculi ⋆

GianLuigi Ferrari

Dipartimento di Informatica, Università di Pisa, giangi@di.unipi.it

Ugo Montanari

Computer Science Laboratory, SRI International, Menlo Park, ugo@csl.sri.com

Abstract

When concurrency is a primitive notion, models of process calculi usually include commuting diamonds and observations of causality links or of abstract locations. However it is still debatable if the existing approaches are natural, or rather if they are an *ad hoc* addition to the more basic interleaving semantics. In the paper a treatment of concurrent process calculi is proposed where the same operational and abstract concurrent semantics described in the literature now descend from general, uniform notions. More precisely we introduce a tile-based semantics for located CCS and we show it consistent with the ordinary concurrent (via permutation of transitions) and bisimilarity based location semantics. Tiles are rewrite rules with side effects, reminiscent of both Plotkin SOS and Meseguer rewriting logic rules. We argue that the tile model is particularly well suited for defining directly operational and abstract semantics of concurrent process calculi in a compositional style.

1 Introduction

Process calculi are usually equipped with notions of operational semantics based on transition systems and of abstract semantics based on observed actions and bisimilarity. Sometimes it is convenient to consider concurrency as a primitive notion, rather than to reduce it to nondeterminism via interleaving. To this purpose, ordinary transition systems have been extended in the literature in several ways. From the operational point of view, certain *commuting diamonds* are introduced (see e.g. [15,4]), whose role is to define as concurrent

⋆ Research supported by Office of Naval Research Contracts N00014-95-C-0225 and N00014-96-C-0114 and by the Information Technology Promotion Agency, Japan, as part of the Industrial Science and Technology Frontier Program "New Models for Software Architechture" sponsored by NEDO (New Energy and Industrial Technology Development Organization). Also research supported in part by CNR Integrated Project *Metodi e Strumenti per la Progettazione e la Verifica di Sistemi Eterogenei Connessi mediante Reti di Comunicazione*; and Esprit Working Groups *CONFER2* and *COORDINA*. The second author is on leave from Dipartimento di Informatica, Pisa, Italy.

those pairs of events which can occur in any order. Concurrent abstract semantics is defined instead by decorating actions with causality links or with abstract locations and possibly by introducing specialized versions of bisimulation [12,11,3,21,9,29,27]. However, while concurrent semantics of process calculi has been given a remarkable attention in the past several years, it is still debatable if the existing approaches are natural, or rather if they are an *ad hoc* addition to the more basic interleaving semantics. We believe a more natural treatment of concurrency should be possible, as we feel has been achieved (at least from an operational point of view) for other models of computations, like Petri nets [13] and term [25,22], graph [6], and term graph [8] rewriting, where axioms generating commuting diamonds are automatically imposed by the framework of definition.

The aim of this paper is to propose a treatment of concurrent process calculi where the same operational and abstract concurrent semantics described in the literature now descend from general, uniform notions. Our approach is based on the *tile model* [17–19]. The tile model relies on certain rewrite rules with side effects, called *tiles*, reminiscent of both SOS rules [30] and rewriting logic rules [25]. Also related models[1] are SOS *contexts* [23] and *structured transition systems* [10].

Tiles have been used for coordination formalisms equipped with flexible synchronization primitives [28,5] and for calculi for mobile processes, like the asynchronous π-calculus [16]. The main advantage of the tile model for handling concurrent process calculi is to integrate a distributed representation of agents and a partial order representation of observations within an SOS-like compositional framework. In particular, with respect to the location approach of [3] the tile version has the advantage of employing only local names and of avoiding infinite branching. Tiles are naturally equipped with a bisimulation-based equivalence relation, which yields the correct notion of process bisimilarity.

We now briefly introduce the tile model. A tile has the form:

$$s \xrightarrow[b]{a} s'$$

and states that the *initial configuration* s of the system evolves to the *final configuration* s' producing an *effect* b. However s is in general *open* (not closed) and the rewrite step is actually possible only if the subcomponents of s also evolve producing the *trigger* a. Both trigger and effect are called *observations*, and model the interaction, during a computation, of the system being described with its environment. More precisely, both system configurations are equipped with an *input* and an *output interface*, and the trigger just describes the evolution of the input interface from its initial to its final configuration. Similarly for the effect. It is convenient to visualize a tile

[1] While tiles can be considered as a generalization of SOS inference rules, their algebraic structure is new. Larsen and Xinxin contexts [23] are analogous, but their algebraic structure is limited to ordinary terms and not axiomatized. Structured transition systems and rewriting logic have similar aims and similar algebraic structure, but do not account for side effects and synchronization.

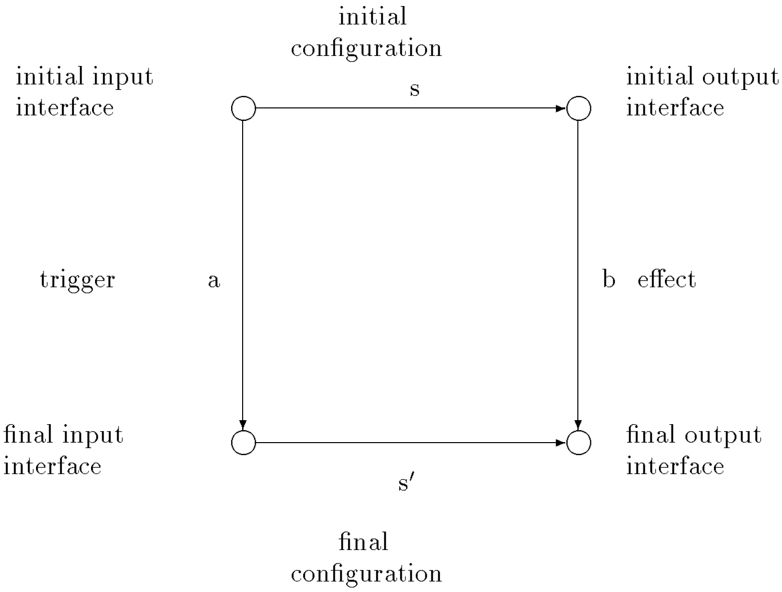

Fig. 1. A tile.

as a two-dimensional structure (see Fig. 1), where the horizontal dimension corresponds to the extension of the system, while the vertical dimension corresponds to the extension of the computation. Actually, we should also imagine a third dimension (the thickness of the tile), which models parallelism: configurations, observations, interfaces and tiles themselves are all supposed to consist of several components in parallel.

To match the SOS style as much as possible and to make more readable the notation, we will more often use the form:

$$\frac{a}{s \xrightarrow{b} s'}$$

Both configurations and observations are assumed to be equipped with operations of parallel and sequential composition (represented by infix operators $\otimes$ and ; respectively) which allow us to build more parallel and larger components, extended horizontally for the configurations and vertically for the observations. Similarly, tiles themselves possess three operations of composition [2]: parallel ($\_\otimes\_$), horizontal ($\_*\_$), and vertical composition. If we consider tiles as logical sequents, it is natural to define the three operations via inference rules called *composition* rules (see Definition 3.2).

The operation of parallel composition is self explanatory. Vertical composion models sequential composition of transitions and computations. Horizontal composition corresponds to synchronization: the effect of the first tile acts as trigger of the second tile, and the resulting tile expresses the synchronized behavior of both. Computing in a *tile logic* consists of starting from a set of basic tiles called *rewrite rules* (and from a set of *auxiliary* tiles which depend

[2] In general, tiles are also equipped with proof terms which distinguish between sequents with the same configurations and observations, but derived in different ways. Suitable axioms for normalizing proof terms are also provided [17–19].

on the version of the tile model at hand), and of applying the composition rules in all possible ways.

A tile logic also can be seen as a double category [14] and tiles themselves as double cells. The categorical interpretation [17,18] is useful since it makes the model more general (configurations and observations can be arrows of any category), allows for universal constructions (e.g. a tile logic is the double category freely generated by its rewrite rules) and suggests analogies with fruitful concepts of algebraic semantics, like institutions. However, the tile model is presented here in a purely logical form.

In this paper, observations and configurations are *term graphs* [2] and term cographs respectively. Term graphs are similar to terms, but two term graphs may explicitly share some of their subterms. Thus in a term graph it is in general not allowed to copy the shared subterms to make the two terms disjoint, since this would yield a different term graph. An axiomatization of term graphs by means of gs-monoidal theories has been recently proposed by Corradini and Gadducci [7,8], and it is reported in the Appendix. Term cographs are like term graphs, but their direction is inverted: while term graphs are oriented from leaves to roots, term cographs are visited from roots to leaves. Term graphs are convenient structures for modeling configurations of distributed systems and their partial ordering observations, since they are equipped with an operation of parallel composition (which models indipendent juxtaposition) and with the possibility of sharing subcomponents. Sharing is used within configurations for modeling the operator $\_ \mid \_$ of process algebras, which in this context means sharing *the same location*. Within observations, sharing is used to express the fact that two events share *the same cause*, or, equivalently, that the same location has two different sublocations.

For instance, the CCS agent $a.nil \mid b.nil$ is represented by the term cograph $G = \{!(a(e)), !(b(e))\}$. The shared variable e represents the common location (the only one in this case) of the two components $!(a(e))$ and $!(b(e))$. Component $!(a(e))$ has one variable but has no root, since the *discharger* operator $!(\_)$ disposes of the result of the subterm $a(e)$. Thus both component $!(a(e))$ and term graph G are arrows from the underlined natural number $\underline{0}$ (i.e. zero roots) to the natural number $\underline{1}$ (i.e. one variable). Term cographs initiating from 0 represent closed agents, and in fact the discharger operator represents the agent nil. Notice that while garbage collection is automatic in term algebra (e.g. $!(a(e)) = !(e)$ since both members represent the empty tuple of terms), in the algebra of term graphs a term graph with no root may carry nontrivial information. Notice also that variables have only local meaning, i.e. in $G = \{!(a(e)), !(b(e))\}$ variable e just represents the only existing variable. In other words also $\{!(a(e')), !(b(e'))\}$ would denote the same term graph G. Only the *ordering* of variables is meaningful.

The computations $a.nil \mid b.nil \xrightarrow{a} nil \mid b.nil \xrightarrow{b} nil \mid nil$ and $a.nil \mid b.nil \xrightarrow{b}$

$a.nil \mid nil \xrightarrow{a} nil \mid nil$ both correspond to the same tile with empty trigger[3]:

$$\alpha:\ !(a(e)), !(b(e)) \xrightarrow{e':=a(e),e'':=b(e),e} !(e), !(e'), !(e'').$$

The effect $e' := a(e), e'' := b(e), e$ of α is a term graph with one variable e and three roots e', e'' and e, i.e. it is an arrow from $\underline{1}$ to $\underline{3}$. Its meaning in terms of events is as follows. At the beginning there is only an initial event e. After the computation, we still have the same initial event, but also two new events e', e'' have happened, labelled by a and b respectively, and both caused by e. In terms of locations, we can say that two sublocations e' and e'' of the initial location e have been created by actions a and b. However, there is no "left" or "right" location, and new locations are introduced only when something takes place at them. The meaning of the final configuration $!(e), !(e'), !(e'')$ is that all the components in the three locations are inactive. A detailed derivation of the above tile is shown in Section 5.

We now show a rewrite rule of our system:

$$(\texttt{Prefix}_\mu)\ \frac{e'}{e' := \mu(e) \xrightarrow{e':=\mu(e),e} e', !(e).}$$

It represents the firing of a prefix μ and corresponds to the following SOS axiom for located CCS [3]:

$$\mu.p \xrightarrow[l]{\mu} l :: p.$$

Notice that $\texttt{Prefix}_\mu$ does not describe the evolution of a closed system, as it is the case for the SOS axiom, since its initial configuration is a partial system $e' := \mu(e) : \underline{1} \to \underline{1}$. However only a trivial (identity) trigger e' is required for applying the rule. A rule with this effect is available in our logic for any agent p and called $id_{[\![p]\!]}$. Thus the tile relevant for agent $\mu.p$ is the horizontal composition $id_{[\![p]\!]} * \texttt{Prefix}_\mu$. This tile describes the creation of a new location, where the final configuration $[\![p]\!]$ is positioned, and states that there is no nontrivial component left at the initial location. Notice that our tile is deterministic while the SOS axiom yields an infinite branching.

The paper is organized as follows. Section 2 introduces term graphs, and the operations of parallel and sequential composition on them. Section 3 presents the tile model, in the simplified version needed in the paper, while Section 4 describes located CCS in a strong version (at our knowledge original) and in the weak version. For the operational semantics of the strong version, the commuting diamonds are defined following the axiomatic approach of [15]. Section 5 defines a tile rewrite system for both the strong and the weak versions, and Section 6 outlines its equivalence with the ordinary semantics of Section 4. In particular, it is shown that the equality of computations specified by the commuting diamonds holds in the tile model, and that its uniform notion of bisimilarity yields in the weak case the same equivalence on agents as ordinary location bisimilarity.

[3] In all our tile rewrite systems modeling process algebras, the tiles with a closed agent as initial configuration have empty trigger, i.e. they can be considered as transitions of a labelled transition system.

2 Term Graphs

In this section we review *term graphs* on which it is based the data stucture we use for modeling configurations and observations. Term graphs [2,1,7,8], have a nice algebraic structure and can be finitely axiomatized as gs-monoidal theories [17,18,7,8]. We follow a style of presentation similar to [1].

Definition 2.1 *(one-sorted term graphs)*
Let us consider a one-sorted, ranked signature Σ, with $f_h \in \Sigma_h$. Furthermore let V be a totally ordered (by $\leq$) infinite set of *names*, a name being denoted by n and similar letters. A *term graph* is a triple $G = (S, var, rt)$ where:

- S is a finite set of sentences, which are assignments of the form $n := f_h(n_1, \ldots, n_h)$, or of the form $n := n'$. In addition, every name must be assigned at most once and no cycles (with the obvious meaning) must be present;
- var is a list, without repetitions, of the *variables* of G, variables being both all the names which are not assigned in S and possibly other names which do not appear in S. Variables are usually denoted by v or similar letters;
- rt is a list, without repetitions, of the *roots* of G, i.e. the names which appear as left members of assignments of the form $n := n'$. Roots are usually denoted by r or similar letters.

The ordering of variables and roots must respect the ordering in V. Furthermore, term graphs are defined up to isomorphic renaming.

Given a term graph G, let h (resp. k) be the number of variables (roots). Then G can be seen as an arrow of *type* $\underline{h} \to \underline{k}$. We write $G : \underline{h} \to \underline{k}$ and call $\underline{h}$ and $\underline{k}$ the *source* and *target* of G respectively. □

For instance, given the signature Σ with $\Sigma_1 = f, g$ and $\Sigma_2 = d$, and the names $n_1, \ldots, n_4, v_1, \ldots, v_3, r_1, r_2$, let $G : \underline{3} \to \underline{2} = (S, var, rt)$ with $S = \{r_1 := v_1, r_2 := n_1, n_1 := d(n_2, n_3), n_2 := g(n_3), n_3 := f(v_1), n_4 := f(v_2)\}$, $rt = (r_1, r_2)$ and $var = (v_1, v_2, v_3)$.

Notice that if we consider as standard the names in the lists of variables and roots (i.e. $v_1, \ldots, v_h$ and $r_1, \ldots, r_k$), isomorphic renaming is restricted to the names which are neither variables nor roots. Furthermore, in this case a term graph is fully specified when its type and its set of sentences are given, since the length of its list of variables can be recovered from its type.

Particularly interesting are the following term graphs which are called *atomic*:
generators: for every $f_h \in \Sigma_h$

$$f_h : \underline{h} \to \underline{1} = \{r_1 := f_h(v_1, \ldots, v_h)\}$$

identities: $id_{\underline{h}}$

$$id_{\underline{h}} : \underline{h} \to \underline{h} = \{r_1 := v_1, \ldots, r_h := v_h\};$$

permutations: $\rho_{\underline{h},\underline{k}}$

$$\rho_{\underline{h},\underline{k}} : \underline{h+k} \to \underline{h+k}$$
$$= \{r_1 := v_{h+1}, \dots, r_k := v_{h+k}, r_{k+1} := v_1, \dots, r_{k+h} := v_h\};$$

duplicators: $\nabla_{\underline{h}}$

$$\nabla_{\underline{h}} : \underline{h} \to \underline{2h} = \{r_1 := v_1, \dots, r_h := v_h, r_{h+1} := v_1, \dots, r_{2h} := v_h\};$$

dischargers: $!_{\underline{h}}$

$$!_{\underline{h}} : \underline{h} \to \underline{0} = \{\}.$$

We now introduce two operations on term graphs. The *sequential composition* of two term graphs is obtained by gluing the list of roots of the first graph with the list of variables of the second, and it is defined only if their numbers are equal. The *parallel composition* instead is always defined, and it is a sort of disjoint union where variable and root lists are concatenated.

Definition 2.2 *(sequential and parallel composition of term graphs)*
Given two term graphs $G_1 : \underline{h} \to \underline{k} = (S_1, var_1, rt_1)$ and $G_2 : \underline{k} \to \underline{l} = (S_2, var_2, rt_2)$, let us take two instances in their isomorphism classes such that $rt_1 = var_2$ and that no other names are shared between G_1 and G_2. Furthermore, let S be the set of clauses in S_1 of the form $r := n$ and let σ be the corresponding name substitution. The *sequential composition* of G_1 and G_2 is the term graph $G_1; G_2 : \underline{h} \to \underline{l} = ((S_1 \setminus S) \cup S_2\sigma, var_1, rt_2)$.

Given two term graphs $G_1 : \underline{h_1} \to \underline{k_1} = (S_1, var_1, rt_1)$ and $G_2 : \underline{h_2} \to \underline{k_2} = (S_2, var_2, rt_2)$, let us take two instances in their isomorphism classes such that no names are shared between G_1 and G_2. The *parallel composition* of G_1 and G_2 is the term graph $G : \underline{h_1 + h_2} \to \underline{k_1 + k_2} = (S_1 \cup S_2, var_1\ var_2, rt_1\ rt_2)$. $\square$

We need a concise term-like notation for term graphs, without explicit lists of variables and roots. Thus we introduce a notion of *presentation* of a term graph which allows for several shorthands. To eliminate the need of specifying variables and roots, we consider the partial ordering on names defined by the assignments. It is definitely true that all roots are maxima and all variables which appear in the assignments are minima. However there might be:
i) maxima which are not roots (like name n_4 in the our example G); and
ii) variables which do not appear at all (like v_3).
To get precisely variables as minima and roots as maxima, we introduce a fictitious mame $\top$ and we assume $n \sqsubseteq \top$ for all names n in i) and ii) above. To express these new dependencies in presentations, we introduce a new sentence $!(n)$ for each such dependency (e.g. $!(n_4)$ and $!(v_3)$ in our example). Moreover, we express the orderings of minima and maxima (which are needed since minima and maxima are now variables and roots) simply by the orderings of their names in V. A second shorthand consists of replacing with its definition every name which is neither a variable nor a root, and which is used just once. For instance in our example we replace $r_2 := n_1$, $n_1 := d(n_2, n_3)$ and $n_2 := g(n_3)$ with $r_2 := d(g(n_3), n_3)$. Finally, we replace assignments like $r_1 := v_1$ in our example, whose only role is to mark as a root a name which is used in other sentences, with sentences consisting of single names, like v_1.

However we must be careful at this point, since the ordering of r_1 among the roots may be different than the ordering of v_1.

Since several presentations may correspond to the same term graph, we give here a reduction procedure able to translate presentations of term graphs into the form of Definition 2.1.

Definition 2.3 *(presentations of term graphs and reduction procedure)*

Given a signature Σ and a set V of names, a *presentation* P of a term graph is a set of sentences of the form n, or $n := T(n_1, \ldots, n_h)$, or $!(T(n_1, \ldots, n_h))$, where T is a term on the signature, possibly just a name. As in Definition 2.1, a name can be assigned at most once and no cycles must be present.

From presentation P we derive as follows a triple (S, var, rt) defining a term graph:

- we replace sentence n with $n' := n$, where n' is a new name with $n \leq n'$ but where no name n'' exists in P with[4] $n \leq n'' \leq n'$;
- we decompose in the obvious way the sentences of the form $n := T(n_1, \ldots, n_h)$, or $!(T(n_1, \ldots, n_h))$ into basic sentences of the form $n := f_h(n_1, \ldots, n_h)$ and $!(n)$, always using new names;
- let P' be the resulting presentation. We define a partial ordering $\sqsubseteq$, where $n \sqsubseteq n'$ iff $n' := n \in P'$ or $n' := f_k(n_1, \ldots, n, \ldots, n_k) \in P'$; and $n \sqsubseteq \top$ iff $!(n) \in P'$, with $\top$ an additional element in the ordering;
- var is the list of the minima in $\sqsubseteq$ ordered according to the total ordering of V. Similarly, rt is the list of the maxima, excluding for $\top$;
- S consists of the set of sentences in P' which are assignments, i.e. sentences $!(n)$ are disregarded.

□

A term-like presentation of our example G is as follows:
$\{n_1, n_3 := d(g(n_2), n_2), n_2 := f(n_1), !(f(n_4)), !(n_5)\}$.

Our notation for presentations is consistent with the ordinary record notation for terms and substitutions, and coincides with it when no sentences n are present, terms like $!(T(n_1, \ldots, n_h))$ are disregarded and no names are left besides variables and roots. Term like presentations will be used throughout the paper. However we should translate our presentations to the basic notation whenever we want to check if two presentations represent the same term graph.

We close this section with some basic properties of term graphs [7,8].

Theorem 2.4 (decomposition of term graphs)

Every term graph can be obtained by evaluating an expression containing only atomic term graphs as constants, and sequential and parallel composition as operators. □

[4] If this is not possible, an α-conversion should be applied to P. These conditions on n' are dictated by the need of yielding the same root ordering for the derived term graph independently from the choice of n'.

For instance the term graph G of our previous example can be represented as:
$G = (\nabla_{\underline{1}}; (id_{\underline{1}} \otimes (f; \nabla_{\underline{1}}; g \otimes id_{\underline{1}}; d))) \otimes (f; !_{\underline{1}}) \otimes !_{\underline{1}}$.

The following theorem gives a characterization of term graphs as *gs-monoidal theories*. A gs-monoidal theory is a logical theory similar to, but weaker than, the algebraic theory of terms and substitutions.

Theorem 2.5 (characterization of term graphs)

The term graphs on the signature Σ *are the arrows of the* gs-monoidal theory $\mathbf{GS}(\Sigma)$ *generated by* Σ. □

In the Appendix we give the finitary axiomatization of gs-monoidal theories presented in [17,18,7,8].

3 The Tile Model

We now describe the basic features of the *tile model*, in the version where observations are term graphs and configurations are term cographs. Term cographs are term graphs where the arrows of the types are all reversed [5].

The presentation follows [18], but is simpler, since the tile sequents we have here (the *flat* sequents) have no associated proof terms. In the following we will call them simply tile sequents.

Definition 3.1 *(tile sequent, tile rewrite system)*

Let Σ_h and Σ_v be two signatures, called the *horizontal* and the *vertical* signature respectively.

A Σ_h-Σ_v *tile sequent* is a quadruple $s \xrightarrow[b]{a} t$, where $s : \underline{h} \to \underline{k}$ and $t : \underline{l} \to \underline{m}$ are term cographs on Σ_h, while $a : \underline{h} \to \underline{l}$ and $b : \underline{k} \to \underline{m}$ are term graphs on Σ_v. (Co)graphs s, t, a and b are called respectively the *initial configuration*, the *final configuration*, the *trigger* and the *effect* of the tile. Trigger and effect are called observations. Underlined integers $\underline{h}$, $\underline{k}$, $\underline{l}$ and $\underline{m}$ are called respectively the *initial input interface*, the *initial output interface*, the *final input interface* and the *final output interface*.

A *tile rewrite system* (TRS) $\mathcal{R}$ is a triple $\langle \Sigma_h, \Sigma_v, R \rangle$, where R is a set of Σ_h-Σ_v sequents called *rewrite rules*. □

A TRS $\mathcal{R}$ can be considered as a logical theory, and new sequents can be derived from it via certain inference rules.

Definition 3.2 *(tile logic)*

Let $\mathcal{R} = \langle \Sigma_h, \Sigma_v, R \rangle$ be a TRS and let $\mathbf{GS}^{op}(\Sigma_h)$ (resp. $\mathbf{GS}(\Sigma_v)$) be the cographs (resp. the graphs) on the signature Σ_h (resp. Σ_v).

Then we say that $\mathcal{R}$ *entails* the class $\mathbf{R}$ of the *tile sequents* $s \xrightarrow[b]{a} t$ obtained by finitely many applications of the following inference rules:

[5] The example of Section 2, $G = \{n_1, n_3 := d(g(n_2), n_2), n_2 := f(n_1), !(f(n_4)), !(n_5)\}$, can be represented as term graph with $(\nabla_{\underline{1}}; (id_{\underline{1}} \otimes (f; \nabla_{\underline{1}}; g \otimes id_{\underline{1}}; d))) \otimes (f; !_{\underline{1}}) \otimes !_{\underline{1}} : \underline{2} \to \underline{3}$ and as term cograph with $((id_{\underline{1}} \otimes (d; g \otimes id_{\underline{1}}; \nabla_{\underline{1}}; f)); \nabla_{\underline{1}}) \otimes (!_{\underline{1}}; f) \otimes !_{\underline{1}} : \underline{3} \to \underline{2}$.

basic rules:

$$(generators)\ \frac{s \xrightarrow[b]{a} t \in R}{s \xrightarrow[b]{a} t \in \mathbf{R}}$$

$$(h\text{-}refl)\ \frac{s : \underline{h} \to \underline{k} \in \mathbf{GS}^{op}(\Sigma_h)}{id_s = s \xrightarrow[id_{\underline{k}}]{id_{\underline{h}}} s \in \mathbf{R}} \qquad (v\text{-}refl)\ \frac{a : \underline{h} \to \underline{k} \in \mathbf{GS}(\Sigma_v)}{id_a = id_{\underline{h}} \xrightarrow[a]{a} id_{\underline{k}} \in \mathbf{R}};$$

composition rules:

$$(p\text{-}comp)\ \frac{\alpha = s \xrightarrow[b]{a} t,\ \alpha' = s' \xrightarrow[b']{a'} t' \in \mathbf{R}}{\alpha \otimes \alpha' = s \otimes s' \xrightarrow[b \otimes b']{a \otimes a'} t \otimes t' \in \mathbf{R}}$$

$$(h\text{-}comp)\ \frac{\alpha = s \xrightarrow[c]{a} t,\ \alpha' = s' \xrightarrow[b]{c} t' \in \mathbf{R}}{\alpha * \alpha' = s; s' \xrightarrow[b]{a} t; t' \in \mathbf{R}}$$

$$(v\text{-}comp)\ \frac{\alpha = s \xrightarrow[b]{a} r,\ \alpha' = r \xrightarrow[b']{a'} t \in \mathbf{R}}{\alpha \cdot \alpha' = s \xrightarrow[b;b']{a;a'} t \in \mathbf{R}};$$

auxiliary rules: (permutations)

$$\rho^{0,0}_{\underline{h},\underline{k}} = \rho_{\underline{h},\underline{k}} \xrightarrow[id_{\underline{h+k}}]{\rho_{\underline{h},\underline{k}}} id_{\underline{h+k}} \in \mathbf{R} \qquad \rho^{0,1}_{\underline{h},\underline{k}} = \rho_{\underline{k},\underline{h}} \xrightarrow[\rho_{\underline{h},\underline{k}}]{id_{\underline{h+k}}} id_{\underline{h+k}} \in \mathbf{R}$$

$$\rho^{1,0}_{\underline{h},\underline{k}} = id_{\underline{h+k}} \xrightarrow[id_{\underline{h+k}}]{\rho_{\underline{k},\underline{h}}} \rho_{\underline{h},\underline{k}} \in \mathbf{R} \qquad \rho^{1,1}_{\underline{h},\underline{k}} = id_{\underline{h+k}} \xrightarrow[\rho_{\underline{k},\underline{h}}]{id_{\underline{h+k}}} \rho_{\underline{k},\underline{h}} \in \mathbf{R}.$$

□

Basic rules provide the sequents corresponding to rewrite rules, together with suitable identity tiles, whose intuitive meaning is that an element of $\mathbf{GS}^{op}(\Sigma_h)$ can be rewritten to itself using only trivial trigger and effect. Similarly for $\mathbf{GS}(\Sigma_v)$. Composition rules provide all the possible ways in which sequents can be composed, while auxiliary rules are the counterpart of the atomic permutation graphs discussed above for term graphs.

For instance the tile denoted $\rho^{0,1}_{\underline{h},\underline{k}}$ consists of a horizontal permutation on the *initial configuration* (notice the character 0 as the *first* upper index) of the tile, and of the inverse permutation on the *effect observation* (notice the character 1 as the *second* upper index). The remaining sides are identities, and similarly for the other permutation tiles. While for reasons of symmetry we include four permutation rules, it is easy to see that one would be enough, since the remaining three could be derived by concatenating one of them with identity tiles.

The role of permutation tiles is to permute the names on one vertex of the tile (the initial output interface in the example), still mantaining the same connections between the adjacent term (co)graphs. For instance, given any tile $\alpha = s \xrightarrow[b]{a} t$ with s and b having $\underline{h+k}$ as target and source respectively, the composition $(id_s * \rho^{0,1}_{\underline{k},\underline{h}}) \cdot (\alpha * id_b)$ produces the tile $\alpha' = s; \rho_{\underline{h},\underline{k}} \xrightarrow[\rho_{\underline{k},\underline{h}};b]{a} t$. Here two permutations have been introduced, but the connections between the two involved term (co)graphs, represented by the composition $s; \rho_{\underline{h},\underline{k}}; \rho_{\underline{k},\underline{h}}; b$, are still the same as $s; b$, since $\rho_{\underline{h},\underline{k}}; \rho_{\underline{k},\underline{h}} = id_{\underline{h+k}}$.

It is easy to see that, by horizontal and vertical composition of the basic permutation tiles, it is possible to obtain permutation tiles $\rho_1 \xrightarrow[\rho_2]{\rho_3} \rho_4$ with arbritary permutations ρ_1, ρ_2, ρ_3 and ρ_4 on the four sides, provided that $\rho_1; \rho_2 = \rho_3; \rho_4$.

It is straightforward to extend the notion of bisimilarity to deal with tile rewrite systems.

Definition 3.3 *(tile bisimilarity).*

Let $\mathcal{R} = \langle \Sigma_h, \Sigma_v, R \rangle$ be a TRS. A symmetric equivalence relation $\equiv_b \subseteq \mathbf{GS}^{op}(\Sigma_h) \times \mathbf{GS}^{op}(\Sigma_h)$ is a *tile bisimulation* for $\mathcal{R}$ if, whenever $s \equiv_b t$ for generic s, t elements of $\mathbf{GS}^{op}(\Sigma_h)$, then for any sequent $\alpha = s \xrightarrow[b]{a} s'$ entailed by $\mathcal{R}$ there exists a corresponding one $\beta = t \xrightarrow[b]{a} t'$ with $s' \equiv_b t'$. The maximal tile bisimulation equivalence is called *tile bisimilarity*, and denoted by $\sim_t$. $\square$

Notice that this notion of bisimilarity is more general than the ordinary one, since it applies to pairs of system components which are open, while the ordinary notion applies only to closed agents.

4 Concurrent and Located Semantics for CCS

There are many concurrent models for process calculi. Some of them focus on the *operational* aspects, defining certain *concurrent machines* for the calculi. Other models are equipped with notions of observation able to capture causal dependencies or localities, and define *abstract semantics*, usually via bisimulation. In this section we try to combine both aspects by presenting a version of concurrent CCS with locations, of which we provide both the concurrent operational and the abstract location semantics. This calculus is supposed to be close to those presented in the literature [15,4,3,21,9,27].

The basic idea of location semantics [3] is to associate a different location with each process, to allow the external observer to see an action together with the location where it takes place. Hence, processes $a.b.nil + b.a.nil$ and $a.nil \mid b.nil$ are distinguished as the second process can perform a and b in different places, while the first process cannot.

To define the concurrent operational semantics, we follow the approach of [15] which associates an n-ary operator to each SOS rule for CCS with n-premises, and then imposes certain χ *axioms* on the resulting algebra of transitions and computations. Concurrent computations are equivalence classes in this formal system. In [20] it is proved that the same equivalence is induced by mapping CCS into Petri nets. It is usually conjectured that the same equivalence can also be derived by following the approach based on proved transition systems and residuals [4].

We first show a strong version of the operational semantics, where locations are visible also in the case of synchronization. We then present a different synchronization rule which hides locations. Besides being closer to the tile version, this presentation of the locality transition system allows us to avoid defining two different kinds of transitions (i.e. *standard* and *location* transitions) as in [3].

Syntax

Let Δ be the alphabet for basic actions, (which is ranged over by α) and $\overline{\Delta}$ the alphabet of complementary actions ($\Delta = \overline{\overline{\Delta}}$ and $\Delta \cap \overline{\Delta} = \emptyset$); the set $\Lambda = \Delta \cup \overline{\Delta}$ will be ranged over by λ. Let $\tau \notin \Lambda$ be a distinguished action, and let $\Lambda \cup \{\tau\}$ (ranged over by μ) be the set of CCS actions.

Let *Loc* be a totally ordered denumerable set of *locations* (ranged over by l). Labels of transitions consist of actions and strings of locations, denoted by u. A synchronization transition is labelled by two strings (in the strong version) or none (in the weak case). The generic denotation is k. In lk, location l is concatenated with each string in k.

The syntax of finite CCS agents with locations is defined by the following grammar.

$$p \ := \ nil \ \Big| \ \mu.p \ \Big| \ l :: p \ \Big| \ p + p \ \Big| \ p \mid p$$

In the following we will consider only agents where p in $\mu.p$ does not contain locations. For the sake of simplicity we do not include restriction, although there is no problem with it. Recursion could also be handled introducing specialized tiles, as in [16].

We use $loc(p)$ and $loc(k)$ to indicate the set of location names occurring in process p or in label k. A process p is called *pure* if $loc(p) = \emptyset$. Throughout the paper we assume that a process contains at most one occurrence of each location. This restriction does not appear in [3]. In [21], however, it has been pointed out that no discriminating power is added if we are allowed to choose a location twice in a computation and that our definition is equivalent to the one in [3].

Operational Semantics

The *Transition Algebra* (TA) of CCS with locations is displayed in Table 1.

For instance, the term $([a, l_1, p >\rfloor p') \mid [\overline{a}, l_2, q >$ describes the proof of the transition:

$$(a.p \mid p') \mid \overline{a}.q \xrightarrow[l_1,l_2]{\tau} (l_1 :: p \mid p') \mid l_2 :: q.$$

As another example, a synchronization of process $l_1 :: a.nil \mid l_2 :: l_3 :: \overline{a}.nil$ is described by the following transition:

$$l_1 :: [a, l_4, nil >\mid l_2 :: l_3 :: [\overline{a}, l_5, nil >:$$

$$l_1 :: a.nil \mid l_2 :: l_3 :: \overline{a}.nil \xrightarrow[l_1l_4,l_2l_3l_5]{\tau} l_1 :: l_4 :: nil \mid l_2 :: l_3 :: l_5 :: nil.$$

We can now introduce the concurrency relation χ.

$$(Act) \quad [\mu, l, p >: \mu.p \xrightarrow[l]{\mu} l :: p$$

$$(Loc) \quad \frac{t : p \xrightarrow[k]{\mu} q, l \notin loc(k)}{l :: t : l :: p \xrightarrow[lk]{\mu} l :: q}$$

$$(Sum) \quad \frac{t : p \xrightarrow[k]{\mu} p'}{t < +q : p + q \xrightarrow[k]{\mu} p'} \qquad \frac{t : p \xrightarrow[k]{\mu} p'}{q+ > t : q + p \xrightarrow[k]{\mu} p'}$$

$$(Comp) \frac{t : p \xrightarrow[k]{\mu} p', loc(k) \cap loc(q) = \emptyset}{t\rfloor q : p \mid q \xrightarrow[k]{\mu} p' \mid q} \quad \frac{t : p \xrightarrow[k]{\mu} p', loc(k) \cap loc(q) = \emptyset}{q\lfloor t : q \mid p \xrightarrow[k]{\mu} q \mid p'}$$

$$(Synch) \frac{t_1 : p_1 \xrightarrow[u_1]{\lambda} q_1, t_2 : p_2 \xrightarrow[u_2]{\overline{\lambda}} q_2, loc(u_1) \cap loc(q_2) = \emptyset = loc(u_2) \cap loc(q_1)}{t_1 \mid t_2 : p_1 \mid p_2 \xrightarrow[u_1, u_2]{\tau} q_1 \mid q_2}$$

Table 1 Transition Algebra

Definition 4.1 *(concurrency relation χ)*

Let $(-$ then $- \chi -$ then $-)$ be a quaternary relation on transition proof terms, defined as the least commutative[6] relation defined by the strucural rules of Table 2, with $t_i : p_i \xrightarrow[k_i]{\mu_i} q_i$, $t'_i : p'_i \xrightarrow[k'_i]{\mu'_i} q'_i$, $t : p \xrightarrow[k]{\mu} q$. □

The concurrency relation χ identifies the *diamonds* in the transition algebra. The axiom defines the basic diamonds, while the inductive rules reproduce the diamonds in all possible contexts. A diamond then forces an identification in the algebra of computations[7]: (t_1 then $t_2 \, \chi \, t_3$ then t_4) implies $t_1; t_2 = t_3; t_4$.

Abstract Semantics

The transition algebra for the abstract semantics is obtained by replacing rules (Act) and $(Synch)$ in Table 1 with the following rules:

$$(Act') \quad [\lambda, l, p >: \lambda.p \xrightarrow[l]{\lambda} l :: p \qquad [\tau, p >: \tau.p \xrightarrow{\tau} p$$

$$(Synch') \frac{t_1 : p_1 \xrightarrow[u_1 l_1]{\lambda} q_1, t_2 : p_2 \xrightarrow[u_2 l_2]{\overline{\lambda}} q_2}{t_1 \mid t_2 : p_1 \mid p_2 \xrightarrow{\tau} d(q_1, l_1) \mid d(q_2, l_2)}$$

where $d(p, l)$ deletes l, i.e. replaces $l :: p'$ with p' in p.

---

[6] Namely, (t_1 then $t_2 \, \chi \, t_3$ then t_4) iff (t_3 then $t_4 \, \chi \, t_1$ then t_2).

[7] A transition is a computation, and composition _;_ of computations is associative and has identities.

$$t_1\rfloor p_2 \text{ then } q_1\lfloor t_2 \,\chi\, p_1\lfloor t_2 \text{ then } t_1\rfloor q_2 \qquad \frac{t_1 \text{ then } t_2 \,\chi\, t_3 \text{ then } t_4}{l :: t_1 \text{ then } l :: t_2 \,\chi\, l :: t_3 \text{ then } l :: t_4}$$

$$\frac{t_1 \text{ then } t_2 \,\chi\, t_3 \text{ then } t_4}{t_1 < +p \text{ then } t_2 \,\chi\, t_3 < +p \text{ then } t_4} \qquad \frac{t_1 \text{ then } t_2 \,\chi\, t_3 \text{ then } t_4}{p+ > t_1 \text{ then } t_2 \,\chi\, p+ > t_3 \text{ then } t_4}$$

$$\frac{t_1 \text{ then } t_2 \,\chi\, t_3 \text{ then } t_4}{t_1\rfloor p' \text{ then } t_2\rfloor p' \,\chi\, t_3\rfloor p' \text{ then } t_4\rfloor p'} \qquad \frac{t_1 \text{ then } t_2 \,\chi\, t_3 \text{ then } t_4}{p'\lfloor t_1 \text{ then } p'\lfloor t_2 \,\chi\, p'\lfloor t_3 \text{ then } p'\lfloor t_4}$$

$$\frac{t_1 \text{ then } t_2 \,\chi\, t_3 \text{ then } t_4}{t_1 \mid t \text{ then } t_2\rfloor q \,\chi\, t_3\rfloor p \text{ then } t_4 \mid t} \qquad \frac{t_1 \text{ then } t_2 \,\chi\, t_3 \text{ then } t_4}{t \mid t_1 \text{ then } q\lfloor t_2 \,\chi\, p\lfloor t_3 \text{ then } t \mid t_4}$$

$$\frac{t_1 \text{ then } t_2 \,\chi\, t_3 \text{ then } t_4, t_1' \text{ then } t_2' \,\chi\, t_3' \text{ then } t_4'}{t_1 \mid t_1' \text{ then } t_2 \mid t_2' \,\chi\, t_3 \mid t_3' \text{ then } t_4 \mid t_4'}$$

Table 2 The Concurrency Relation

Using $(Synch')$, the synchronization of process $l_1 :: a.nil \mid l_2 :: l_3 :: \overline{a}.nil$ of the previous example becomes:

$$l_1 :: a.nil \mid l_2 :: l_3 :: \overline{a}.nil \xrightarrow{\tau} l_1 :: nil \mid l_2 :: l_3 :: nil$$

as defined in [3].

We now introduce the notion of location bisimilarity. We adopt the standard notation for weak transitions: $\overset{\epsilon}{\Rightarrow} = (\xrightarrow{\tau})^*$ and $\underset{u}{\overset{\lambda}{\Rightarrow}} = \overset{\epsilon}{\Rightarrow} \underset{u}{\overset{\lambda}{\rightarrow}} \overset{\epsilon}{\Rightarrow}$.

Definition 4.2 *(location bisimilarity)*

A binary relation $\mathcal{R}$ is a location simulation if $p\mathcal{R}q$ implies:

- for each $p \underset{ul}{\overset{\lambda}{\Rightarrow}} p'$, with $loc(l) \cap loc(q) = \emptyset$, there exists some $q \underset{ul}{\overset{\lambda}{\Rightarrow}} q'$ with $p'\mathcal{R}q'$;
- for each $p \overset{\epsilon}{\Rightarrow} p'$ there exists some $q \overset{\epsilon}{\Rightarrow} q'$ with $p'\mathcal{R}q'$.

A relation $\mathcal{R}$ is called a location bisimulation if both $\mathcal{R}$ and $\mathcal{R}^{-1}$ are location simulations. Two processes p and q are *location bisimilar* ($p \approx_l q$) if $p\mathcal{R}q$ for some location bisimulation $\mathcal{R}$. □

5 A Tile Rewrite System for Concurrent Located CCS

The aim of this section is to show how the framework provided by the tile model can be applied to provide natural concurrent, located semantics for CCS.

In what follows we introduce the components of the tile rewrite system, i.e. horizontal signature, vertical signature and rewrite rules.

Horizontal Signature.

The symbols of the signature Σ_h, and their arities, are as follows:

$\mu\ : 1\ (Prefix)$

$+ : 1\ (Choice)$

$\overleftarrow{+} : 1\ (Left - Choice)$

$\overrightarrow{+} : 1\ (Right - Choice)$

$\kappa\ : 0\ (Codischarger).$

Configurations of the tile rewrite system are term cographs over the signature Σ_h.

Vertical Signature

The symbols of Σ_v are as follows

$\mu\ : 1\ (Action)$

$T : 2\ (Synchronization)$

$\overleftarrow{T} : 1\ (Left - T)$

$\overrightarrow{T} : 1\ (Right - T)$

Observations are term graphs over the signature Σ_v.

In the above signature, symbols $+$, $\overleftarrow{+}$ and $\overrightarrow{+}$ are used to translate the CCS operator $+$. Similarly, T, $\overleftarrow{T}$ and $\overrightarrow{T}$ model the τ action obtained via synchronization. Symbol κ is used for making inactive processes refused in a choice.

Since configurations and observations of rules are term (co)graphs[8], we use for them the notation developed in Section 2. We denote names as e, e', etc.

Strong Rewrite Rules

To match the SOS notation as closely as possible, from now on a tile $s \xrightarrow[b]{a} s'$ will be represented as:

$$\frac{a}{s \xrightarrow{b} s'}$$

Following the SOS convention, the antecedent (trigger) a will be omitted when it is the empty term graph.

The rewrite rules are displayed in Table 3. Notice that roots (variables) of term cographs representing configurations belong to input (output) interfaces

[8] Employing gs-monoidal theories equipped with hypersignatures would allow for replacing $+$, $\overleftarrow{+}$, and $\overrightarrow{+}$ with a unique symbol $+'$. Similarly, a symbol T' could replace T, $\overleftarrow{T}$ and $\overrightarrow{T}$.

$$(\texttt{Prefix}_\mu)\ \frac{e'}{e' := \mu(e) \xrightarrow{e':=\mu(e),e} e', !(e)}$$

$$(\texttt{Suml}_\mu)\ \frac{e_1, e_2, e' := \mu(e_1)}{e_1 :=\overleftarrow{+}(\overline{e}), e_2 :=\overrightarrow{+}(\overline{e}), \overline{e} := +(e) \xrightarrow{e_1:=e, e':=\mu(e)} e', e_1, e_2 := \kappa}$$

$$(\texttt{Comp}_\mu)\ \frac{e, e_1, e' := \mu(e)}{e, e_1 := e \xrightarrow{e, e':=\mu(e)} e, e', e_1 := e}$$

$$(\texttt{Synch}_\lambda)\ \frac{e_1, e_2, e'_1 := \lambda(e_1), e'_2 := \overline{\lambda}(e_2)}{e_1, e_2 \xrightarrow{e'_1:=\overleftarrow{T}(e'), e'_2:=\overrightarrow{T}(e'), e':=T(e_1,e_2), e_1, e_2} e_1, e'_1, e_2, e'_2}$$

$$(\texttt{Twin})\ \frac{e'_1 :=\overleftarrow{T}(e'), e'_2 :=\overrightarrow{T}(e'), e' := T(e_1, e_2), e_1, e_2}{e_1 := e, e_2 := e \xrightarrow{e'_1:=\overleftarrow{T}(e'), e'_2:=\overrightarrow{T}(e'), e':=T(e,e), e} e_1 := e, e_2 := e, e'_1, e'_2}$$

Table 3 Strong Rewrite System

of the tiles, while variables (roots) of term graphs representing observations belong to initial (final) interfaces of the tiles.

We can now comment on the definition of the rules. The application of the rule $\texttt{Prefix}_\mu$ causes the rewriting of the initial configuration $e' := \mu(e)$ into the final configuration $e', !(e)$ where a new variable has been created. The intuition is that this new variable corresponds to the name of the event associated to the firing of the prefix. Such a name is never cancelled by further rewriting steps. The rule ($\texttt{Comp}_\mu$) basically describes the asyncronous evolution of parallel processes (those associated to e_1 and e). To illustrate the application of this rule let us consider the rewriting of process $a.nil \mid b.nil$.

We start with the $\texttt{Prefix}_\texttt{a}$ rule:

$$\frac{e'}{e' := a(e) \xrightarrow{e':=a(e),e} e', !(e)}$$

we compose horizontally $id_{!_{\underline{1}}}$ with it, and the result in parallel with $id_{!_{\underline{1}};b}$. We thus get:

$$!(a(e)), !(b(e_1)) \xrightarrow{e':=a(e),e,e_1} !(e), !(e'), !(b(e_1)).$$

We are now ready to compose this tile horizontally with $\texttt{Comp}_\texttt{a}$:

$$\alpha = !(a(e)), !(b(e)) \xrightarrow{e':=a(e),e} !(b(e)), !(e').$$

We can now compose horizontally $id_{!_{\underline{1}}}$ with $\mathtt{Prefix_b}$, and the result in parallel with $id_{!_{\underline{1}}}$:

$$!(b(e)), !(e') \xrightarrow{e'':=b(e),e,e'} !(e), !(e'), !(e'').$$

Finally, by composing vertically α with the latter tile:

$$!(a(e)), !(b(e)) \xrightarrow{e':=a(e),e'':=b(e),e} !(e), !(e'), !(e'').$$

The effect of this tile (the term graph $e' := a(e), e'' := b(e), e$) tells us that the two actions can be executed in parallel.

Rule $\mathtt{Synch}_\lambda$ accounts for synchronizations at different locations, like $a.b.nil \mid c.\overline{b}.nil \xrightarrow{ac\tau} nil \mid nil$. Rule $\mathtt{Twin}$, when $\mathtt{Synch}_\lambda$ is composed horizontally with it, allows for synchronizations at one location only, like $a.nil \mid \overline{a}.nil \xrightarrow{\tau} nil \mid nil$.

Rule $\mathtt{Suml}_\mu$ puts a codischarger constant κ on the refused branch. Refused alternatives will appear as inactive $\otimes$ factors in configurations.

In addition to $\mathtt{Comp}_\mu$, we also have rules $\mathtt{TlComp}_\mu$, $\mathtt{TrComp}_\mu$ and $\mathtt{TwinComp}$ which take care of composition for T-moves (left and right side) and twin T-moves. We also have a rule $\mathtt{Sumr}_\mu$. We do not show these rules.

According to our definition of term graph given in Section 2, an ordering of variables and roots must be provided. For instance, it is essential in computing the sequential composition of two term graphs. Thus it would appear as necessary to specify the ordering of names in all the interfaces of our tiles. For instance, what is the ordering between e' and e_1 in the final input interface of rule $\mathtt{Comp}_\mu$? However it is easy to see that name ordering is immaterial for rules. In fact, given a rule α it is always possible to obtain tiles with different orderings of names in the interfaces by composing α with suitable auxiliary (permuation) tiles. Thus any consistent renaming in the presentation [9] of the rules of a tile rewrite system $\mathcal{R}$ does not change the tiles entailed by $\mathcal{R}$.

Abstract Semantics

We would like to handle weak location bisimulation with the uniform notion of tile bisimulation presented in Definition 3.3. To this purpose, it is necessary to add rewrite rules able to transform effects with τ and T observations into identities.

More precisely, we need an additional symbol in the horizontal signature:

$F : 1 \;\; (\mathit{Filter})$

[9] Remember that names appear only in our *presentations* for term graphs and tiles. Tiles themselves contain no names.

and the following three additional *weak* rules:

$$(\texttt{Filter}_\lambda) \qquad \frac{e_1, e_1' := \lambda(e_1)}{e_1 := F(e) \xrightarrow{e,e':=\lambda(e)} e_1 := F(e), e_1' := F(e')}$$

$$(\texttt{Filter}_\tau) \qquad \frac{e_1, e_1' := \tau(e_1)}{e_1 := F(e) \xrightarrow{e} e_1 := e', e_1' := e', e' := F(e)}$$

$$(\texttt{Filtersynch}) \ \frac{e_1'' := \overleftarrow{T}(e'), e_2'' := \overrightarrow{T}(e'), e' := T(e_1', e_2'), e_1', e_2'}{e_1' := F(e_1), e_2' := F(e_2) \xrightarrow{e_1,e_2} G}$$

where $G = \{e_1' := \overline{e_1}, e_1'' := \overline{e_1}, e_2' := \overline{e_2}, e_2'' := \overline{e_2}, \overline{e_1} := F(e_1), \overline{e_2} := F(e_2)\}$.

The *weak rewrite system* consists of the strong rules and of the above three weak rules.

The filters work as follows. If we start from a configuration with a filter for every variable, observations containing λ are able to go through as they are, while τ and T observations are transformed into identities. Notice that in the latter case the new name(s) generated by the transition (one for τ and two for T) are merged with the names before the transition, i.e. eliminated. Notice also from $\texttt{Filter}_\lambda$ that filters reproduce themselves on every newly generated variable.

Location bisimilarity for our configurations is thus defined for the weak rewrite system as tile bisimulation (see Definition 3.3).

6 Comparing SOS and Tile Semantics

We first show the correspondence for the concurrent operational semantics and then for the abstract location semantics.

Operational Semantics

To show the correspondence of our tile rewrite system with the transition algebra normalized with the concurrency relation χ, we first translate located agents p into configurations $[\![p]\!]$. It is convenient to define inductively the auxiliary function $ind[p]$ as returning a pair of configurations, the first translating the sequential components of p which do not contain a location, the second referring to the located components. Function $[\![p]\!]$ is the parallel composition of the two configurations, where the variables corresponding to the located components are ordered according to the total ordering of *Loc*.

Definition 6.1 *(from located CCS agents to configurations)*

Let p be a located CCS agent. Then $[\![p]\!] : \underline{0} \to \underline{|loc(p)| + 1}$ is the configuration corresponding to agent p, where function $[\![\_]\!]$ is defined below. In the inductive definition we assume $ind[p] = \langle f, g\rangle$, where $f : \underline{0} \to \underline{1}$ and $g : \underline{0} \to \underline{|loc(p)|}$, and similarly for p_1 and p_2.

$$
\begin{aligned}
ind[nil] &= \langle !_{\underline{1}}, id_{\underline{0}}\rangle\\
ind[\mu.p] &= \langle f;\mu\ ,\ id_{\underline{0}}\rangle\\
ind[p_1 + p_2] &= \langle f_1 \otimes f_2; \overleftarrow{+} \otimes \overrightarrow{+}; \nabla_{\underline{1}}; +\ ,\ id_{\underline{0}}\rangle\\
ind[l :: p] &= \langle !_{\underline{1}}, f \otimes g\rangle\\
ind[p_1 \mid p_2] &= \langle f_1 \otimes f_2; \nabla_{\underline{1}}\ ,\ g_1 \otimes g_2\rangle\\
[\![p]\!] &= f \otimes (g; \sigma_p)
\end{aligned}
$$

where $\sigma_p = \{r_1 := v_{i_1}, \ldots, r_n := v_{i_n}\}$, with $i_1, \ldots, i_n$ the list of locations of p ordered according to the *prefix tree walk* of p, and $v_l < v_{l'}$ iff $l < l'$. □

Maybe the only surprising clause is that for $l :: p$. It states that, when an agent is prefixed with a location, the component which is nonlocated becomes the first located component, while the new nonlocated component is empty. Notice that this ordering of located components (as the one for $p_1 \mid p_2$) is consistent with the prefix tree walk of p.

Some examples, where we use our notation for term graphs, should make the mapping clear. We start with some pure agents.

$$
\begin{aligned}
&[\![a.b.nil]\!] = !(b(a(e)))\\
&[\![a.nil \mid b.nil]\!] = !(a(e)), !(b(e))\\
&[\![(a.nil \mid b.nil) \mid (c.nil + d.nil)]\!] =\\
&!(a(e_1)), !(b(e_1)), e_1 := e, !(c(\overleftarrow{+}\ (e_2))), !(d(\overrightarrow{+}\ (e_2))), e_2 := +(e) =\\
&!(a(e)), !(b(e)), !(c(\overleftarrow{+}\ (e_2))), !(d(\overrightarrow{+}\ (e_2))), e_2 := +(e).
\end{aligned}
$$

Considering now a located agent $p = l_1 :: (l_7 :: a.nil \mid l_5 :: b.nil)$ we have:

$$[\![p]\!] = !_{\underline{1}} \otimes !_{\underline{1}} \otimes (!_{\underline{1}}; a) \otimes (!_{\underline{1}}; b); id_{\underline{2}} \otimes \rho_{1,1}\ \ : \underline{0} \to \underline{4}$$

or, using our term graph notation:

$$[\![p]\!] = !(e_0), !(e_1), !(b(e_2)), !(a(e_3)).$$

Notice that variable e_0 is not used in $[\![p]\!]$, since there is no sequential component in p which is nonlocated, while we also have $!(e_1)$ since no sequential component is prefixed only by l_1.

As another example, we have:

$$[\![a.nil \mid (b.nil \mid l :: c.nil)]\!] = !(a(e_0)), !(b(e_0)), !(c(e_1)).$$

In what follows, we will consider configurations defined up to $\otimes$ factors $s = [\![p]\!]; \kappa$. Such components are inactive, in the sense that any tile α having $s \otimes s'$ as initial configuration can be decomposed as $\alpha = id_s \otimes \alpha'$, where α' is a tile having s' as initial configuration.

It is possible to define a translation $\{\!|\_|\!\}$ from transitions to tiles derivable in the strong rewrite system, using induction on transition proof terms. The tiles obtained in this way are complete, in the sense that, when composed vertically, they yield essentially all derivable tiles. Furthermore, given any diamond of the concurrency relation χ of Table 2, the translated computations commute.

Proposition 6.2 (from proved transitions to tiles)

It is possible to define inductively a function $\{\!|t|\!\}$ from terms t of the transition algebra in Table 1 to tiles such that the following properties hold.

(i) *Tile $\{\!|t|\!\}$ is derivable in the strong tile rewrite system.*

(ii) *Any derivable tile with initial configuration $[\![p]\!]$ can be obtained by repeatedly composing vertically tiles in the range of $\{\!|\_|\!\}$ and tiles composed of identity and auxiliary tiles.*

(iii) *A diamond (t_1 then $t_2 \, \chi \, t_3$ then t_4) implies $\{\!|t_1|\!\} \cdot \{\!|t_2|\!\} = \{\!|t_3|\!\} \cdot \{\!|t_4|\!\}$.*

□

We now briefly outline the proof of Proposition 6.2. As hinted above, function $\{\!|\_|\!\}$ is defined by induction on transition proof constructors. Clauses for (Act), (Loc) and (Sum) are easy:
$\{\!|\ [\mu, l, p>\ |\!\} = id_{[\![p]\!]} * \mathtt{Prefix}_\mu$;
$\{\!|l :: t|\!\} = id_{!_{\underline{1}}} \otimes \{\!|t|\!\}$;
$\{\!|t < +p|\!\} = (\{\!|t|\!\} \otimes id_{[\![p]\!]}) * \mathtt{Suml}_\mu$.

We can also hint at the form of the clauses for ($Comp$) and ($Synch$), even if writing them in detail may require the development of suitable notation. For $\{\!|t\rfloor q|\!\}$ we have several cases. If q has no nonlocated component, or if the action of t takes place at located components only (either one or two according to the action being μ or T) then an identity is enough. Otherwise, if t performs an action μ then rule $\mathtt{Comp}_\mu$ is used. If t performs a synchronization between two nonlocated components, rule $\mathtt{TwinComp}$ is needed. Finally, if t performs a synchronization between a non located and a located component, rules $\mathtt{TlComp}_\mu$ or $\mathtt{TrComp}_\mu$ are required.

For $\{\!|t_1 \mid t_2|\!\}$ we always use $\mathtt{Synch}_\lambda$. We also use $\mathtt{Twin}$ if the synchronization involves nonlocated components for both p_1 and p_2, $\mathtt{TlComp}_\mu$ or $\mathtt{TrComp}_\mu$ if only one of them is located, nothing else otherwise. In all cases we need permutation tiles built from auxiliary tiles $\rho^{i,j}_{\underline{h},\underline{k}}$ to ensure suitable "wire twisting". For instance, using permutation tiles it is possible to bring close the components we want to synchronize, to synchronize them, and to bring them back to the original positions.

It is possible to see that derivable tiles starting from $[\![p]\!]$ can be broken down vertically into *one-step tiles*, i.e. tiles computed by $\{\!|\_|\!\}$, and *perm-tiles* of the form $s \xrightarrow{\rho} s; \rho$, where ρ is any permutation. This decomposition is essential for the correctness of the tile semantics shown in the next paragraph.

The validity of the χ axiomatization in the tile model can be easily checked on the inductive definition of $\{\!|\_|\!\}$. It would be interesting if the χ axiomatization were also complete, i.e. enough to equate all computations in the CCS algebra yielding the same tile via $\{\!|\_|\!\}$. This is not the case here, since e.g.

the two transitions of $a.nil + a.nil$ would yield the same tile but would not be equated by χ. A natural option to try for capturing exactly the operational concurrency of located CCS would be to consider tiles equipped with a proof term as originally defined in [18].

Abstract Semantics

Location bisimilarity and tile location bisimilarity coincide for pure CCS agents.

Proposition 6.3 (tile semantics is correct)
Given two pure CCS agents p and q, we have[10] $p \approx_l q$ *iff* $[\![p]\!]; F \sim_t [\![q]\!]; F$.$\Box$

We sketch the proof of Proposition 6.3.

- We define a transition algebra with a slightly more informative label. Instead of $p \xrightarrow[k]{\mu} q$ we require transitions $p \xrightarrow{o} q = p \xrightarrow[loc(p),k]{\mu} q$, exposing also the locations of the agent. It is easy to see that this modification does not change bisimilarity.
- We define a function[11] $[\![o]\!]$ on transition labels as follows:

$$\begin{aligned} [\![\tfrac{\tau}{L}]\!] &= id_{\underline{|L|}}; \\ [\![\tfrac{\mu}{\{l_1,\ldots,l_n\},l}]\!] &= \{l := \mu(l_0), l_0, l_1, \ldots, l_n\}; \\ [\![\tfrac{\mu}{\{l_1,\ldots,l_n\},ul_il}]\!] &= \{l := \mu(l_i), l_0, l_1, \ldots, l_n\}. \end{aligned}$$

  Notice that our function depends only on the last two locations (i.e. it does not depend on u). In fact, it has been noticed [27] that the same bisimilarity relation is obtained relying on an *incremental* observation, where the string of locations is truncated to the last two items.
- We define modified functions $[\![\_]\!]$ and $\{\!|\_|\!\}$ (for which however we will use the same denotations as in the strong case) to replace SOS rules (Act) and $(Synch)$ with (Act') and $(Synch')$ and to include filters and weak rewrite rules. Taking advantage of Proposition 6.2, we then show that if the initial configuration and the effect of a tile are $[\![p]\!]$ and $[\![o]\!]$ for some p and o, then the tile is the vertical composition of one-step tiles.
- We show that if $p \approx_l q$ and $[\![p]\!] \xrightarrow{a} s_1$, $[\![q]\!] \xrightarrow{a} s_2$ are perm-tiles, then there exist p' and q' with $p' \approx_l q'$ such that $[\![p']\!] = s_1$ and $[\![q']\!] = s_2$. Informally, this is true because permuting the variables in the observation corresponds just to apply some injective substitution to agent locations, which does not affect bisimilarity.
- To show the *only if* part, given a bisimulation S for configurations, we prove that if we take pRq for agents whenever $[\![p]\!]S[\![q]\!]$, then R is also a bisimulation. To this purpose, we show that given a pair pRq and a transition $t : p \xrightarrow{o} p'$, we have $\{\!|t|\!\} : [\![p]\!] \xrightarrow{[\![o]\!]} [\![p']\!]$. Thus there is a tile $[\![q]\!] \xrightarrow{[\![o]\!]} s$ with

[10] Remember that F is the filter needed to convert strong effects into weak effects.
[11] We overload the notation used for mapping configurations, since the meaning is analogous.

$[\![p']\!]Ss$. This tile consists of the vertical composition of one-step tiles. As a consequence we have $q \stackrel{o}{\Rightarrow} q'$ with $[\![q']\!] = s$ and $p'Rq'$.

- To show the *if* part, we prove that if we take $[\![p]\!];\rho S[\![q]\!];\rho$ for every permutation ρ whenever $p \approx_l q$, then S is also a bisimulation. In fact, given a pair $[\![p]\!]S[\![q]\!]$, a tile $[\![p]\!] \stackrel{a}{\longrightarrow} s$ can be either a one-step tile or a perm-tile (or a vertical composition of several of these tiles, but this case is as usual subsumed by the shorter moves). If it is a perm-tile with permutation ρ, then also $[\![q]\!]$ has a perm-tile with the same ρ and the final configurations are in S since $[\![p]\!];\rho S[\![q]\!];\rho$ by construction. If $[\![p]\!] \stackrel{a}{\longrightarrow} s$ is a one-step tile, then there exist o and p' with $p \stackrel{o}{\longrightarrow} p'$, $[\![o]\!] = a$ and $[\![p']\!] = s$. Thus we have a computation $q \stackrel{o}{\Rightarrow} q'$ with $p' \approx_l q'$. As a consequence, there is a tile $[\![q]\!] \stackrel{a}{\longrightarrow} [\![q']\!]$, which is the vertical composition of the tiles corresponding to the transitions in the computation. Finally the proof obligation for $[\![p]\!];\rho S[\![q]\!];\rho$ is already fulfilled, since by a property previously proved there are agents p'' and q'', with $p'' \approx_l q''$; and $[\![p'']\!] = [\![p]\!];\rho$ and $[\![q'']\!] = [\![q]\!];\rho$.

7 Conclusions

In the paper we have presented a version of the tile model aimed at providing a uniform operational and abstract framework for concurrent process calculi. As a case study, we have presented tile rewrite systems for strong and weak versions of located CCS and we have shown their correctness. An advantage of the tile approach is the full compositionality of the underlying logic, which is able to handle computations of open system components as they were new rewrite rules specifying complex behaviors. Another innovative aspect is related to the use of term graphs for representing distributed configurations and partial ordering observations.

The case study in the paper concerns *located* CCS, but it is easy to see that the strong version presented here is actually the same that is needed for *causal* CCS [11]. The weak causal version however is more complex, since an event can have any number of immediate causes. To handle this case, we should consider more complicate structures then term graphs as observations. For instance we should have, besides a ∇ for sharing causes, also another atomic graph $\overline{\nabla}$ to share effects.

8 Acknowledgments

We would like to thank Roberto Bruni, Fabio Gadducci, Narciso Marti-Oliet and Marco Pistore for their comments.

References

[1] Z.M. Ariola, J.W. Klop, *Equational Term Graph Rewriting*, Fundamenta Informaticae 26, 207–240, 1996.

[BD95] E. Badouel, P. Darondeau, *Trace Nets and Process Automata*, Acta Informatica **32** (7), 1995. pp. 647-679.

[2] H.P. Barendregt, M.C.J.D. van Eekelrn, J.R.W. Glauert. J.R. Kennaway. M.J. Plasmeijer, M.R. Sleep, *Term Graph Reduction*, Proc. PARLE, Springer LNCS 259, 141–158, 1987.

[3] G. Boudol, I. Castellani, M. Hennessy and A. Kiehn, *Observing Localities*, Theoretical Computer Science, 114: 31–61, 1993.

[4] G. Boudol, I. Castellani, *Flow Models of Distributed Computations: Three Equivalent Semantics for CCS*, Information and Computation 114(2):247-314 (1994).

[5] R. Bruni, U. Montanari, *Zero-Safe Nets, or Transition Synchronization Made Simple*, to appear in Proc. Express'97, Santa Margherita, September 1997.

[6] A. Corradini, H. Ehrig, M. Löwe, U. Montanari, F. Rossi, *Abstract Approach to Graph Transformation - Part I: Basic Concepts and Double Pushout Approach*, in: G Rozenberg, Ed., The Handbook of Graph Grammars and Computing by Graph Transformation, Vol.1: Foundations.

[7] A. Corradini, F. Gadducci, *An Algebraic Presentation of Term Graphs via Gs-Monoidal Categories*, submitted for publication. Available at http://www.di.unipi.it/ gadducci/papers/aptg.ps, 1997.

[8] A. Corradini, F. Gadducci, *A 2-Categorical Presentation of Term Graph Rewriting*, Proc. CTCS'97, Springer LNCS, to appear, 1997.

[9] F. Corradini, R. De Nicola, *Distribution and Locality of Concurrent Systems*, in Proc. ICALP'94, LNCS 920, pages 154–165. Springer Verlag, 1994.

[10] A. Corradini, U. Montanari, *An Algebraic Semantics for Structured Transition Systems and its Application to Logic Programs*, Theoretical Computer Science **103**, 1992, pp. 51-106.

[11] P. Darondeau, P. Degano, *Causal Trees*, In: Proc. ICALP 89, LNCS, Vol. 372, 1989.

[12] P. Degano, R. De Nicola, U. Montanari, *Partial Ordering Descriptions and Observations of Nondeterministic Concurrent Processes*, in: Linear Time, Branching Time and Partial Order in Logics and Models for Concurrency, LNCS 354, 1989.

[13] P. Degano, J. Meseguer, U. Montanari, *Axiomatizing the Algebra of Net Computations and Processes*, Acta Informatica Vol.33(7), (October 1996) pp.641-667.

[14] C. Ehresmann, *Catégories Structurées*: I and II, Ann. Éc. Norm. Sup. 80, Paris (1963), 349-426; III, Topo. et Géo. diff. V, Paris (1963).

[15] G. Ferrari, U. Montanari, *Towards the Unification of Models for Concurrency*, in Proc. Colloqium on Trees and Algebra of Programming (CAAP 90), LNCS 431, 162–176, 1990.

[16] G. Ferrari, U. Montanari, *A Tile-Based Coordination View of the Asynchronous π-calculus*, to appear in Proc. MFCS'97, Springer LNCS, 1997.

[17] F. Gadducci, *On the Algebraic Approach to Concurrent Term Rewriting*, PhD Thesis, Università di Pisa, Pisa, Technical Report TD-96-02, Department of Computer Science, University of Pisa, 1996.

[18] F. Gadducci, U. Montanari, *The Tile Model*, Technical Report TR-96-27, Department of Computer Science, University of Pisa, 1996.

[19] F. Gadducci, U. Montanari, *Tiles, Rewriting Rules and CCS*, in Proc. 1st international workshop on Rewriting Logic and Applications, J. Meseguer Ed., ENTCS 4 (1996), pp.1-19.

[20] R. Gorrieri, U. Montanari, *On the Implementation of Concurrent Calculi into Net Calculi: Two Case Studies*, TCS 141, 1-2, 1995 pp.195-252.

[21] A. Kiehn, *Local and Global Causes*, Acta Informatica 31, 697-718 (1994).

[22] C. Laneve, U. Montanari, *Axiomatizing Permutation Equivalence*, Math. Struct. in Comp. Sc. (1996), vol.6, pp.219-249.

[23] K.G. Larsen, L. Xinxin, *Compositionality Through an Operational Semantics of Contexts*, in Journal of Logic and Computation, vol.1, n.6, 1991 (conference version in Proc. ICALP'90, Springer-Verlag, LNCS 443, 1990).

[24] F. W. Lawvere, *Functorial Semantics of Algebraic Theories*, Proc. National Academy of Science **50**, 1963, pp. 869-872.

[25] J. Meseguer, *Conditional Rewriting Logic as a Unified Model of Concurrency*, Theoretical Computer Science **96**, 1992, pp. 73-155.

[26] R. Milner, J. Parrow, D. Walker. *A Calculus of Mobile Processes* (parts I and II), Information and Computation, 100:1–77, 1992.

[27] U. Montanari, M. Pistore, D. Yankelevich, *Efficient Minimization up to Location Equivalence*, in Proc. ESOP'96, LNCS 1058. Springer Verlag, 1996.

[28] U. Montanari, F. Rossi, *Graph Rewriting and Constraint Solving for Modelling Distributed Systems with Synchronization*, in: Paolo Ciancarini and Chris Hankin, Eds., Coordination Languages and Models, LNCS 1061, 1996, pp. 12-27. Full paper submitted for publication.

[29] U. Montanari, D. Yankelevich, *Location Equivalence in a Parametric Setting*, Theoretical Computer Science **149**, 1995.

[30] G. Plotkin, *A Structural Approach to Operational Semantics*, Technical Report DAIMI FN-19, Computer Science Department, Aarhus University, 1981.

Appendix: Gs-Monoidal Theories

Here the interested reader may find the axiomatic definition (taken from [17,18,7,8]) of gs-monoidal theories (in the case of an ordinary signature).

They are similar to the ordinary algebraic (Lawvere) theories [24], the difference being the missing naturality axioms for duplicators and dischargers. Gs-monoidal theories are monoidal theories, since the naturality axiom of permutations holds instead.

Definition .1 *(graphs)* A *graph* G is a 4-tuple $\langle O_G, A_G, \delta_0, \delta_1\rangle$: O_G, A_G are sets whose elements are called respectively *objects* and *arrows* (ranged over by $a, b, \ldots$ and $f, g, \ldots$), and $\delta_0, \delta_1 : A_G \to O_G$ are functions, called respectively *source* and *target*. A graph G is *reflexive* if there exists an *identity* function $id : O_G \to A_G$ such that $\delta_0(id(a)) = \delta_1(id(a)) = a$ for all $a \in O_G$; it is *with pairing* if its class O_G of objects forms a monoid; it is *monoidal* if it is reflexive with pairing and also its class of arrows forms a monoid, such that id, δ_0 and δ_1 respect the neutral element and the monoidal operation. $\square$

Definition .2 *(one-sorted gs-monoidal theories)* Given a signature Σ, the associated *gs-monoidal theory* $\mathbf{GS}(\Sigma)$ is the monoidal graph with objects the elements of the commutative monoid $(\underline{\mathbf{N}}, \otimes, \underline{0})$ (where $\underline{0}$ is the neutral object and the sum is defined as $\underline{n} \otimes \underline{m} = \underline{n+m}$); and arrows those generated by the following inference rules:

$$(generators)\ \frac{f_k \in \Sigma}{f_k : \underline{k} \to \underline{1} \in \mathbf{GS}(\Sigma)} \qquad (sum)\ \frac{s : \underline{n} \to \underline{m}, t : \underline{n'} \to \underline{m'}}{s \otimes t : \underline{n} \otimes \underline{n'} \to \underline{m} \otimes \underline{m'}}$$

$$(identities)\ \frac{\underline{n} \in \underline{\mathbf{N}}}{id_{\underline{n}} : \underline{n} \to \underline{n}} \qquad (composition)\ \frac{s : \underline{n} \to \underline{m}, t : \underline{m} \to \underline{k}}{s; t : \underline{n} \to \underline{k}}$$

$$(duplicators)\ \frac{\underline{n} \in \underline{\mathbf{N}}}{\nabla_{\underline{n}} : \underline{n} \to \underline{n} \otimes \underline{n}} \qquad (dischargers)\ \frac{\underline{n} \in \underline{\mathbf{N}}}{!_{\underline{n}} : \underline{n} \to \underline{0}}$$

$$(permutations)\ \frac{\underline{n}, \underline{m} \in \underline{\mathbf{N}}}{\rho_{\underline{n},\underline{m}} : \underline{n} \otimes \underline{m} \to \underline{m} \otimes \underline{n}}$$

Moreover, the composition operator ; is associative, and the monoid of arrows satisfies the functoriality axiom:

$$(s \otimes t); (s' \otimes t') = (s; s') \otimes (t; t')$$

whenever both sides are defined; the identity axiom: $id_{\underline{n}}; s = s = s; id_{\underline{m}}$ for all $s : \underline{n} \to \underline{m}$; the monoidality axioms:

$$id_{\underline{n} \otimes \underline{m}} = id_{\underline{n}} \otimes id_{\underline{m}} \qquad \rho_{\underline{n} \otimes \underline{m}, \underline{p}} = (id_{\underline{n}} \otimes \rho_{\underline{m},\underline{p}}); (\rho_{\underline{n},\underline{p}} \otimes id_{\underline{m}})$$

$$!_{\underline{n} \otimes \underline{m}} = !_{\underline{n}} \otimes !_{\underline{m}} \qquad \nabla_{\underline{n} \otimes \underline{m}} = (\nabla_{\underline{n}} \otimes \nabla_{\underline{m}}); (id_{\underline{n}} \otimes \rho_{\underline{n},\underline{m}} \otimes id_{\underline{n}})$$

$$!_{\underline{0}} = \nabla_{\underline{0}} = \rho_{\underline{0},\underline{0}} = id_{\underline{0}} \qquad \rho_{\underline{0},\underline{n}} = \rho_{\underline{n},\underline{0}} = id_{\underline{n}}$$

for all $\underline{n}, \underline{m}, \underline{p} \in \underline{\mathbf{N}}$; the *coherence* axioms:

$$\nabla_{\underline{n}}; (id_{\underline{n}} \otimes \nabla_{\underline{n}}) = \nabla_{\underline{n}}; (\nabla_{\underline{n}} \otimes id_{\underline{n}}) \qquad \nabla_{\underline{n}}; \rho_{\underline{n},\underline{n}} = \nabla_{\underline{n}}$$

$$\nabla_{\underline{n}}; (id_{\underline{n}} \otimes !_{\underline{n}}) = id_{\underline{n}} \qquad \rho_{\underline{n},\underline{m}}; \rho_{\underline{m},\underline{n}} = id_{\underline{n}} \otimes id_{\underline{m}}$$

for all $\underline{n}, \underline{m} \in \underline{\mathbf{N}}$; and the *naturality* axiom:

$$(s \otimes t); \rho_{\underline{m},\underline{q}} = \rho_{\underline{n},\underline{p}}; (t \otimes s)$$

for all $s : \underline{n} \to \underline{m}, t : \underline{p} \to \underline{q} \in \mathbf{GS}(\Sigma)$. $\square$